El ahorcado

Layouts, Componentes JLabel y JButton

La interfaz gráfica

La ventana

- Esquema básico de una aplicación Swing.
  - Programa principal
  - Ventana que hereda de Jframe
  - Lámina que hereda de Jpanel

- Pero todavía no muestra nada

```java
package proyectos;

import javax.swing.*;

public class ElAhorcado {

    // Clase para ejecutar la aplicación
    public static void main(String[] args) {
        new ElAhorcado().iniciar();
    }

    private void iniciar() {

    }

    // Ventana
    class Ventana extends JFrame{
        public Ventana(){

        }
    }

    // Panel para contener los elementos de la interfaz
    class LaminaPrincipal extends JPanel{
        public LaminaPrincipal(){

        }
    }
}
```

La ventana

Al iniciar, creamos un objeto de ventana y especificamos que hacer cuando se pulse cerrar ventana.

```
public void iniciar() {
    Ventana ventana=new Ventana();
    ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

En el constructor de la ventana

- Ponemos la posición y el tamaño
- Ponemos el título
- Creamos y añadimos la lámina
- Lo hacemos visible

```
class Ventana extends JFrame{
    public Ventana() {
        setBounds( 350, 300, 600, 500);
        setTitle("Juego del Ahorcado");

        LaminaPrincipal laminaPrincipal=new LaminaPrincipal();
        add(laminaPrincipal);

        setVisible(true);
    }
}
```


Juego del Ahorcado

Componente Etiqueta

- Con el componente JLabel podemos poner texto en nuestras aplicaciones
- Si lo añadimos a nuestra lámina

```
public LaminaPrincipal(){
    JLabel etiqueta=new JLabel( text: "Etiqueta juego del ahorcado");
    add(etiqueta);
}
```

Componente Etiqueta

Constructores

```
JLabel()
JLabel(String text)
JLabel(String text, int horizontalAlignment)
JLabel(String text, Icon icon, int horizontalAlignment)
JLabel(Icon image)
JLabel(Icon image, int horizontalAlignment)
```

Métodos

- setFont (heredado de JComponent)
- setText
- setHorizontalAligment
- setIcon

Componente Etiqueta

```
public LaminaPrincipal() {
    JLabel etiqueta=new JLabel("Etiqueta Juego del ahorcado");
    etiqueta.setFont(new Font("Ubuntu",Font.BOLD,24));
    etiqueta.setIcon(new ImageIcon("imagenes/ahorcado.png"));
    add(etiqueta);
}
```

Componente Botón

Constructores

Los constructores de los botones son parecidos a las etiquetas, también se les puede especificar un icono.

Métodos

- setText y getText
- setIcon
- setEnabled
- ...

Heredados...

- setFont (heredado de JComponent)
- setSize
- setBackground

Problemas

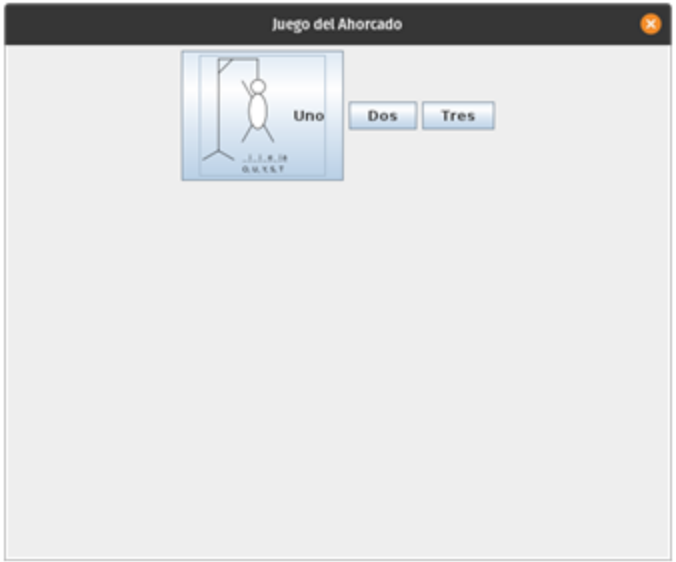
Ahora queremos poner una serie de botones.
Creamos una segunda lámina con 3 botones y la añadimos después del título

```
class LaminaBotones extends JPanel{

    public LaminaBotones() {
        add(new JButton("Uno",new ImageIcon("imagenes/ahorcado.png")));
        add(new JButton("Dos"));
        add(new JButton("Tres"));
    }
}
```

```
        LaminaPrincipal laminaPrincipal=new L
        add(laminaPrincipal);
        add(new LaminaBotones());
        setVisible(true);
```

Pero el resultado no es el esperado, ha desaparecido el título.



El problema, la disposición

La solución, la disposición

DISPOSICIONES (Layouts)

Disposiciones

La disposición en cómo un componente muestra los componentes que contiene

Cada componente tiene una disposición por defecto, pero se puede cambiar.

Tipos de disposiciones

- BorderLayout
- FlowLayout
- GridLayout
- CardLayout
- BoxLayout
- GridBagLayout
- SpringLayout
- GroupLayout
- Sin Layout

A Visual Guide to Layout Managers

Several AWT and Swing classes provide layout managers for general use:

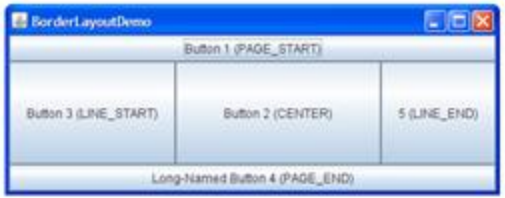
- `BorderLayout`
- `BoxLayout`
- `CardLayout`
- `FlowLayout`
- `GridBagLayout`
- `GridLayout`
- `GroupLayout`
- `SpringLayout`

This section shows example GUIs that use these layout managers, and tells you where to find the how-to page for each layout manager. You can find links for running the examples in the how-to pages and in the example index.

Note: This lesson covers writing layout code by hand, which can be challenging. If you are not interested in learning all the details of layout management, you might prefer to use the `GroupLayout` layout manager combined with a builder tool to lay out your GUI. One such builder tool is the NetBeans IDE. Otherwise, if you want to code by hand and do not want to use `GroupLayout`, then `GridBagLayout` is recommended as the next most flexible and powerful layout manager.

If you are interested in using JavaFX to create your GUI, see Working With Layouts in JavaFX.

BorderLayout



Every content pane is initialized to use a `BorderLayout`. (As Using Top-Level Containers explains, the content pane is the main container in all frames, applets, and dialogs.) A `BorderLayout` places components in up to five areas: top, bottom, left, right, and center. All extra space is placed in the center area. Tool bars that are created using JToolBar must be created within a `BorderLayout` container, if you want to be able to drag and drop the bars away from their starting positions. For further details, see How to Use BorderLayout.

Una guía visual a los gestores de disposiciones

Disposiciones FlowLayout

Es el gestor por defecto de los paneles.

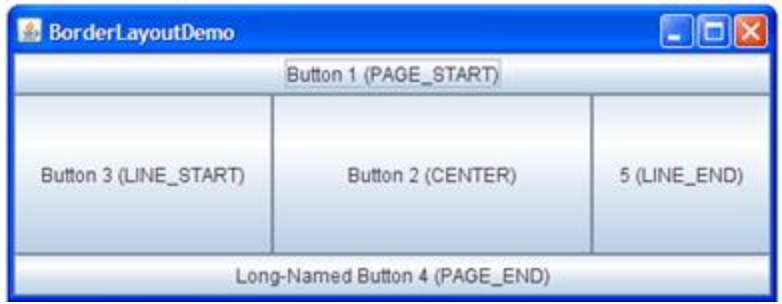
Los componentes se disponen en una fila según el orden en que se añaden. Si no hay espacio se van creando columnas.

Se puede especificar la alineación y la distancia de separación entre los componentes, ya sea en el constructor o mediante métodos.

```
FlowLayout()
FlowLayout(int align)
FlowLayout (int align, int hgap, int vgap)
```

La alineación puede ser FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER

Disposiciones BorderLayout



Es el gestor por defecto de los Frames

Divide el contenedor en 5 zonas, izquierda, derecha, arriba, abajo y centro. Al añadir un componente se especifica en qué zona se añade, si no se especifica la zona se añadirá en el centro.

Si dos componentes se añaden en la misma zona, se visualizará el último añadido.

Cada zona se ajusta al tamaño de los componentes que contiene, si sobra espacio lo coge la zona central

```
BorderLayout()
Constructs a new border layout with r
BorderLayout(int hgap, int vgap)
```

Se puede especificar el espacio entre zonas.

Añadir componente

- add(componente,zona)

Zona puede ser constantes de clase

WEST, EAST, NORTH, SOUTH, CENTER

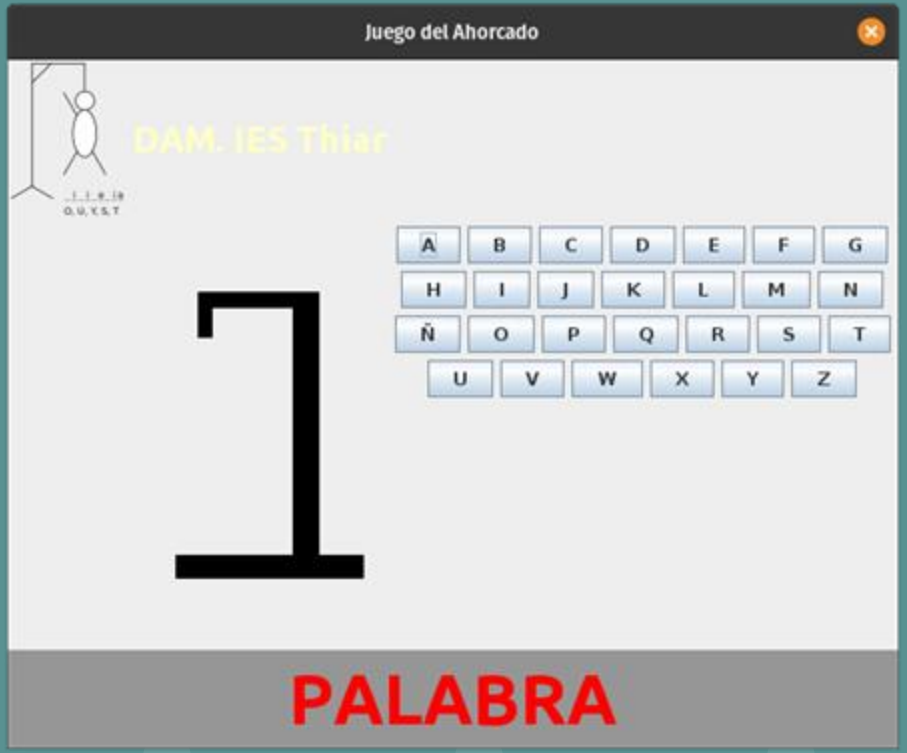
Disposiciones múltiples

Podemos combinar varios elementos en nuestra interfaz y que cada uno de ellos tenga su propia disposición.

En nuestra aplicación queremos un título en la parte superior, mostrar una imagen a la izquierda de nuestra interfaz y mostrar botones a la derecha, para ello podemos usar un BorderLayout. Estos elementos los podemos colocar en el Frame, pero vamos a usar un panel, nuestro panel principal y en el insertar todos los elementos.

Los botones queremos que se muestren uno a continuación del otro, un FlowLayout puede hacer el trabajo perfectamente, por lo que tendremos un panel que es el colocamos en el BorderLayout, y los botones en el panel.

Disposición final



PanelPrincipal

Dejamos la clase de la ventana igual, añadiendo únicamente una instancia de la Lámina Principal y quitando el panel de botones.

En el constructor de LaminaPrincipal cambiamos su layout a BorderLayout y añadimos las etiquetas del título al norte y una para la imagen a la izquierda.

Creamos un panel para añadir una etiqueta que nos servirá para mostrar las pistas de la palabra secreta. Al ponerla en un panel y añadir el panel al sur logramos que la etiqueta esté centrada.

Por último creamos el panel para contener los botones de las letras y mediante un método añadimos los botones.

```
private JLabel imagen,respuesta;
private JPanel panelLetras;
public LaminaPrincipal() {
    setLayout(new BorderLayout());
    JLabel titulo=new JLabel("DAM. IES Thiar");
    titulo.setFont(new Font("Ubuntu", Font.BOLD, 24));
    titulo.setForeground(new Color(255,255,200));
    titulo.setIcon(new ImageIcon("imagenes/ahorcado.png"));
    add(titulo,BorderLayout.NORTH);

    imagen=new JLabel();
    imagen.setIcon(new ImageIcon("imagenes/Hangman-0.png"));
    add(imagen,BorderLayout.LINE_START);

    JPanel panelRespuesta=new JPanel();
    panelRespuesta.setBackground(new Color(150,150,150));
    respuesta=new JLabel("PALABRA");
    respuesta.setFont(new Font("Ubuntu", Font.BOLD, 48));
    respuesta.setForeground(new Color(255,0,0));
    panelRespuesta.add(respuesta);
    add(panelRespuesta,BorderLayout.PAGE_END);

    panelLetras=new JPanel();
    ponerBotones();
    add(panelLetras);

}
```

Añadir los botones

Todos aquellos componentes que vayamos a usar en métodos deben ser accesibles, por eso panelLetras es un atributo encapsulado de la clase.

```
public void ponerBotones() {
    String letras="ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    for (int i=0;i<letras.length();i++) {
        JButton boton=new JButton(""+letras.charAt(i));
        panelLetras.add(boton);
    }
}
```

Funcionalidad

Ocultar las letras de la palabra secreta

Necesitamos una palabra para adivinar. En pruebas vamos a usar un atributo en la clase LaminaPrincipal de tipo cadena de texto, le pongo el nombre *secreto* y le ponemos un valor (THIAR en mi caso)

Mediante una función recorremos las letras de ese atributo y “ocultamos” las letras.

Llamamos a la función dentro del constructor.

```
public void formaLetrasSecretas() {
    String formaSecreto="";
    for(int i=0;i<secreta.length();i++) {
            formaSecreto+="_ ";
    }
    respuesta.setText(formaSecreto.trim());
}
```

El resultado sería

Pulsando los botones
Clase interna gestora

```
class TeclaPulsada implements ActionListener{
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        JButton botonPulsado=(JButton)e.getSource();
        botonPulsado.setEnabled(false);
        comprobarLetra(e.getActionCommand());
    }
}
```

Creamos una clase interna para el manejo de las pulsaciones.

- Obtenemos el botón que se ha pulsado
- Con `setEnabled(false)` lo deshabilitamos para que no se pueda volver a pulsar
- `e.getActionCommand()` nos devuelve el texto del botón, que recordamos tiene la letra
- `comprobarLetra` es un método el método que va a hacer todo el trabajo

Nos falta enlazar el botón con el gestor, en el método que creaba los botones creamos una instancia del gestor y lo enlazamos.

```
public void ponerBotones() {
    String letras="ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    TeclaPulsada gestor=new TeclaPulsada();
    for (int i=0;i<letras.length();i++) {
        JButton boton=new JButton(""+letras.charAt(i));
        boton.addActionListener(gestor);
        panelLetras.add(boton);
    }
}
```

El trabajo duro

Esta es la función que realiza todo el trabajo. Recibe una cadena de caracteres que debe corresponder con una letra.

Añade la letra a una lista de caracteres, letrasPulsadas, que se ha definido como atributo de la clase y es usado para comprobar con las letras de la palabra secreta.

Con fallos se comprueba las veces que se falla y se usa para cambiar la imagen.

Por último se comprueba la finalización, ya sea ganador o perdedor.

```
    private String secreta="THIAR";
    private int fallos=0;
    private final int MAX_FALLOS=6;
    private ArrayList<Character> letrasPulsadas=new ArrayList();
```

```
public void comprobarLetra(String c) {
    boolean acertada=true; // Para saber si la palabra está completa
    letrasPulsadas.add(c.charAt(0)); // Añadimos la letra a la lista
    if (secreta.contains(c)) { // La letra pulsada está en la palabra
        // Ponemos el texto de la etiqueta ocultando las letras
        // que todavía no se han pulsado
        String formaSecreto="";
        for(int i=0;i<secreta.length();i++) {
            if (letrasPulsadas.contains(secreta.charAt(i))) {
                formaSecreto=formaSecreto+secreta.charAt(i)+" ";
            }
            else {
                // Si alguna no está, no está completa la palabra
                acertada=false;
                formaSecreto+="_ ";
            }
        }
        respuesta.setText(formaSecreto.trim());
    }
    else {
        // La letra pulsada no está en la palabra
        fallos++;
        acertada=false;
        // Usamos el número de fallos para cambiar la imagen
        imagen.setIcon(new ImageIcon("imagenes/Hangman-"+fallos+".png"));
    }
    if (acertada) {
        // Se ha acertado la palabra
        System.out.println("Has ganado");
    }
    if (fallos==MAX_FALLOS) {
        // Ahorcado
        System.out.println("AHORCADO");
    }

}
```

Referencias y Bibliografía

- Vídeos del 81 al 85 del curso Java Desde 0 del canal PildorasInformaticas

- API Java
- Guía visual a los gestores Layouts

- Programación avanzada. Interfaces Gráficas de usuario. Oscar Belmonte Fernández
- Programación orientada a objetos con Java usando BlueJ, 6e, por Barnes, David; Kölling, Michael
- Introduction to Java Programming and Data Structures, por Y Daniel Liang

Ejercicios 1

1. El título de la aplicación sale a la izquierda. Como hemos añadido la etiqueta directamente al BorderLayout lo pone a la izquierda. Lo queremos centrado, para conseguirlo debemos añadir la etiqueta a un panel y este panel al norte del BorderLayout. Además personifica el título con tu nombre.

2. Cambia la disposición de los elementos de la aplicación. Intenta simular la siguiente disposición, o puedes inventarte la tuya propia siempre que sea adecuada.

3. Crea una ArrayList con unas cuantas palabras. Elige una al azar como la palabra secreta

Cuadros de diálogo

Dialogos. JOptionPane

API

Ya hemos visto el uso de cuadros de diálogo, hemos usado los métodos estáticos.

`JOptionPane.showMessageDialog` para mostrar mensajes al usuario

`JOptionPane.showInputDialog` para al usuario que introdujera algún dato

Son ventanas **modales**, ésto significa que toman el control de la aplicación, no podemos volver a la aplicación principal hasta no cerrar el diálogo.

JOptionPane.showMessageDialog Mostrando mensajes

Tenemos 3 formas de llamar al métodos según si introducimos 2, 4 o 5 parámetros.

- componentParent, podemos indicarle el componente padre. El diálogo saldrá centrado en él.
- message, indica el mensaje a mostrar
- title, título de la ventana
- messageType, tipo de mensaje ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE. Sirve para cambiar la imagen que aparece.
- icon, para mostrar un icono

Ejemplos:

```
JOptionPane.showMessageDialog(null,”El juego ha finalizado”);

JOptionPane.showMessageDialog(panel, "Lo siento, has perdido",
"AHORCADO",JOptionPane.INFORMATION_MESSAGE);
```

JOptionPane.showInputDialog
Obteniendo datos

Tenemos 6 formas de llamar al métodos según los parámetros que introduzcamos al método.

Devuelve una cadena de texto con los datos que introduce el usuario. Ejemplos:

```
String nombre=JOptionPane.showInputDialog("Introduce tu nombre");
```


```
String nombre=JOptionPane.showInputDialog("Introduce tu nombre ",
"Valor inicial");
```


```
String nombre=JOptionPane.showInputDialog(panel,"Introduce tu nombre",
"Titulo dialogo",JOptionPane.WARNING_MESSAGE);
```

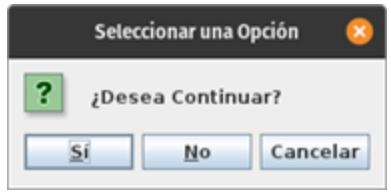
JOptionPane.showConfirmDialog
Preguntas de confirmación

Mediante este cuadro de diálogo preguntaremos al usuario con las respuestas en botones. Los botones pueden ser SI, NO, CANCELAR y con el título por defecto de “Selecciona una opción”. Como los anteriores, tenemos sobrecarga con varios métodos.

El método devuelve un entero que representa el botón pulsado, esos valores están definidos en constantes de JOptionPane (YES_OPTION, NO_OPTION, CANCEL_OPTION)

```
int opcion=JOptionPane.showConfirmDialog(panel, "¿Desea Continuar?");
if (opcion==JOptionPane.YES_OPTION)
        System.out.println("Continuar");
```

```
int opcion=JOptionPane.showConfirmDialog(panel, "¿Desea Continuar?");
if (opcion==JOptionPane.YES_OPTION)
        System.out.println("Continuar");
```



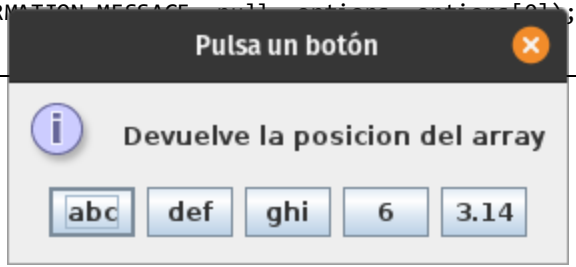
JOptionPane.showOptionDialog
Preguntas de opciones

Mediante este cuadro de diálogo preguntaremos el usuario podrá elegir entre varias opciones. Devuelve la posición del botón pulsado.

Sólo tenemos un método, sin sobrecarga:

```
public static int showOptionDialog(Component parentComponent, Object message, String title, int optionType,
int messageType, Icon icon, Object[] options, Object initialValue) throws HeadlessException
```

```
Object[] options = {"abc", "def", "ghi", 6, 3.14};
int x = JOptionPane.showOptionDialog(null, "Devuelve la posición del array","Pulsa un botón",
JOptionPane.DEFAULT_OPTION,                    JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);
System.out.println(x);
```



Referencias

- Vídeos del 121 y 122 del curso Java Desde 0 del canal PildorasInformaticas

- http://codejavu.blogspot.com/2013/12/ejemplo-joptionpane.html

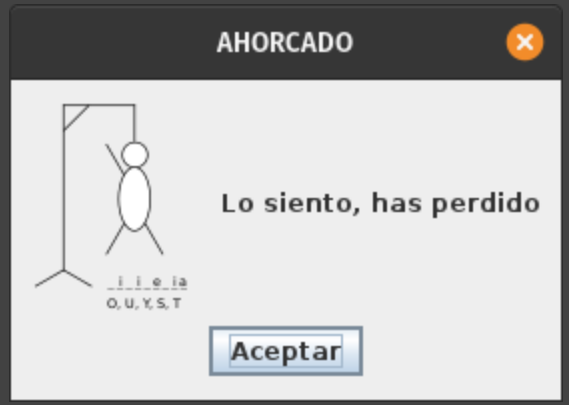
Ejercicios 2

1. Al finalizar el juego, informamos con un cuadro de diálogo si ha ganado o ha perdido.
2. Deshabilitamos todas las teclas.

AYUDA

```
Component[] botonesLetras = panelLetras.getComponents();
```

con getComponents obtenemos todos los componentes de un contenedor, es un array de componentes, y el método setEnabled, como muchos otros, es un método de Component



1. Preguntamos con un diálogo si quiere seguir jugando o no. Si responde si, reiniciamos el juego. Si responde no, cerramos la aplicación.

Eventos de ventana

WindowEvent - WindowListener

En los eventos de ventana se utiliza la interfaz WindowListener que captura eventos del tipo WindowEvent. La interfaz define 7 métodos por lo que es conveniente utilizar su adaptador WindowAdapter:

● windowActivated(WindowEvent e)	Cuando se activa la ventana
● windowClosed(WindowEvent e)	Cuando la ventana se ha cerrado, después del dispose
● windowClosing(WindowEvent e)	Cuando el usuario pide cerrar la ventana
● windowDeactivated(WindowEvent e)	Cuando la ventana se desactiva
● windowDeiconified(WindowEvent e)	Cuando se des minimiza la ventana
● windowIconified(WindowEvent e)	Cuando se minimiza la ventana
● windowOpened(WindowEvent e)	Cuando es la primera vez que la ventana se hace visible

Ejemplo

Si probamos el ejemplo veremos que muestra un mensaje al abrirse la ventana, y cuando la cerramos mostrará el mensaje de cerrando la ventana, pero no el de Ventana cerrada, se destruye la ventana antes de mostrar el mensaje.

Esto es debido a la instrucción

```
setDefaultCloseOperation(EXIT_ON_CLOSE);
```

si la cambiamos por

```
setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
```

Veremos que ahora no nos muestra el mensaje y tampoco se cierra.

Si añadimos en el método windowClosing la instrucción dispose(); después del mensaje, cerramos la ventana, veremos que ahora muestra todos los mensajes.

```
package ahorcado;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class EventosVentana extends JFrame{

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new EventosVentana("Ejemplo Eventos");
    }

    public EventosVentana(String titulo) {
        super(titulo);
        setSize(400, 300);
        setLocationRelativeTo(null);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        addWindowListener(new AdaptadorVentana());
    }

    class AdaptadorVentana extends WindowAdapter{

        public void windowClosing(WindowEvent e) {
            // TODO Auto-generated method stub
            super.windowClosing(e);
            System.out.println("Cerrando la ventana");
        }
        public void windowClosed(WindowEvent e) {
            // TODO Auto-generated method stub
            super.windowClosed(e);
            System.out.println("Ventana cerrada");
        }
        public void windowActivated(WindowEvent e) {
            // TODO Auto-generated method stub
            super.windowActivated(e);
            System.out.println("Ventana activada");
        }

    }
}
```

Ejercicios 2. Reto

1. ¿Podemos hacer que al pulsar sobre el icono de cerrar la ventana nos pregunte si realmente queremos cerrar la aplicación?

AYUDA

Necesitamos que la acción por defecto al cerrar la aplicación sea no hacer nada.

Capturar el evento de cerrando la ventana… y entonces preguntar si queremos cerrar, si la respuesta es sí, cerrar definitivamente la aplicación